Negotiating and delegating obligations

Meriam Ben Ghorbel, Frédéric Cuppens, Nora Cuppens-Boulahia

Negotiating and Delegating Obligations

Meriam Ben-Ghorbel-Talbi
Institut TELECOM/Telecom Bretagne
2 rue de la Châtaigneraie, 35576 Cesson Sévigné Cedex, France
meriam.benghorbel@telecom-bretagne.eu

Frédéric Cuppens
Institut TELECOM/Telecom Bretagne
2 rue de la Châtaigneraie, 35576 Cesson Sévigné Cedex, France
frederic.cuppens@telecom-bretagne.eu

Nora Cuppens-Boulahia
Institut TELECOM/Telecom Bretagne
2 rue de la Châtaigneraie, 35576 Cesson Sévigné Cedex, France
SWID
80 avenue des Buttes de Coesmes, 35700 Rennes, France
nora.cuppens@telecom-bretagne.eu

ABSTRACT

In this paper, we describe a security model where users are allowed to control their obligations partially or totally, depending on the security policy. The main motivation of our work is to design more flexible systems that take into account users' requirements in order to avoid obligation violations and therefore sanctions. In our model, users are able to negotiate or delegate their obligations in the case of incapacity to fulfill them. This is an important aspect to be considered, since it is common that, at work or in everyday life, a user may need to negotiate the fulfillment of a given obligation, or also need the help of others to perform a task on his/her behalf. This may be due to several reasons such as absence, vacation, conflict of interest, lack of time, of resource, of competence or simply for the sake of efficiency. In our model, we propose an approach to deal with the negotiation and the delegation of obligations based on the concept of contexts.

Categories and Subject Descriptors

D.4.6 [**OPERATING SYSTEMS**]: Security and Protection—*Access controls*

General Terms

Security

Keywords

Obligations, Delegation, Context, The OrBAC model

1. INTRODUCTION

Obligations [6, 15, 16] are an important means to specify security control, in particular usage control [19, 20]. Obligations must not be violated and must be fulfilled by a fixed deadline, otherwise sanctions are inflicted upon users. Punitive sanctions may correspond to the activation of new security rules such as prohibitions or new obligations. This means that when users fail to fulfill their obligations, they may have their privileges revoked or a more costly obligations may be prescribed. Sanctions are necessary in order to keep the system compliant with its security policy, but, in some cases, users are not able to fulfill their obligations. For instance, in the case of their absence, if they have another obligation that is conflictual or if they do not have sufficient resource, competence or time to fulfill the requested task. This requires the security models to be more flexible in order to allow users to manage their own obligations to avoid inevitable violations and unjust sanctions. Our main motivation is to propose a system that guarantees that users have sufficient means to fulfill their obligations by their self or by others. This is important to ensure the correct behavior of the system and thus its security, especially when obligations are related to sensitive data, to security tasks or to the availability of services.

For this purpose, we first propose the concept of obligation negotiation. In our model, authorized users are able to negotiate their obligation revision with the responsible entity, called authority, in order to obtain its consent. Negotiation may concern different aspects of the obligation such as the task, the sanction or also the deadline. For instance, in a conference program committee a reviewer can negotiate with the PC Chair the revision deadline in order to extend it. But, negotiation is not always possible or authorized. For instance, governmental obligations such as the payment of tax or traffic offense fine are mandatory and in the case of violation, can impose heavy financial burdens. Therefore, we also propose the use of obligation delegation [10, 18, 23, 3]. For instance, when a user is not available, he/she can delegate the payment of his/her traffic offense fine or electricity bill to another user. Note that in this case one can only delegate the obligation to pay, but not the responsibility and the sanction related to it, e.g. to pay a higher bill or to go to jail. In other cases, users can delegate the obligation and also the responsibility related to it. For this

MEDES'10 October 26-29, 2010, Bangkok, Thailand

reason, we propose to distinguish between these two notions as we suggested in [3]. Moreover, we argue that the consent of the user who will receive the delegation (called the delegatee) must be required, except for some situations where there are power relationships. To this end, we propose a delegation protocol to allow users to negotiate the delegation of obligations and responsibilities. We propose to manage consent requirement using the concept of pre-obligations [5, 13], i.e. the delegation is allowed only if the user has requested and obtained the consent. We also introduce the concept of obligation pre-notification in order to inform users that their obligations will be activated or violated. This allows users to know their obligations (and obligations that are delegated to them) in order to fulfill them on time, to negotiate or also to delegate them.

This paper is organized as follows. In section 2, we start with the system description. We give the main concepts of our model and how we use contexts to deal with the obligation activation and violation. In section 3, we introduce the concept of notification of obligation and violation. Then, in section 4 we discuss obligation negotiation and we present our negotiation protocol. We give in section 5 our delegation protocol and we introduce the concept of dynamic consent and pre-obligations. In section 6, we present related work. Finally, concluding remarks and future work are made in section 7.

2. SYSTEM DESCRIPTION

We are based on a highly structured RBAC model called OrBAC [11]. This model provides means to specify the security policy at the organization level that is independently of the implementation of this policy. Thus, instead of modeling the policy by using the concrete concepts of subject, action and object, the OrBAC model suggests reasoning with the roles that subjects, actions or objects play in the organization. The role of a subject is simply called a role, whereas the role of an action is called an activity and the role of an object is called a view. Moreover, the concept of context is explicitly introduced in OrBAC, which actually provides our model with high flexibility and expressiveness. More details about the OrBAC model and the motivation for using it to deal with delegation are given in [4]. We give hereafter the basic concepts that are required in this paper.

2.1 Basic predicates

In OrBAC, there are eight basic sets of entities: Org (a set of organizations), S (a set of subjects), A (a set of actions), O (a set of objects), R (a set of roles), $\mathcal{A}$ (a set of activities), V (a set of views) and C (a set of contexts). In the following we give the basic OrBAC built-in predicates:

- ***Empower*** is a predicate over domains $Org \times S \times R$. If org is an organization, s a subject and r a role, $Empower(org, s, r)$ means that org empowers s in r.

- ***Use*** is a predicate over domains $Org \times O \times V$. If org is an organization, o is an object and v is a view, then $Use(org, o, v)$ means that org uses o in v.

- ***Consider*** is a predicate over domains $Org \times A \times \mathcal{A}$. If org is an organization, α is an action and a is an activity, then $Consider(org, \alpha, a)$ means that org considers that action α implements activity a.

- ***Hold*** is a predicate over domains $Org \times S \times A \times O \times C$. If org is an organization, s a subject, α an action, o an object and c a context, $Hold(org, s, \alpha, o, c)$ means that within organization org, context c holds between s, α and o.

- ***Permission*** is a predicate over domains $Org_s \times R_s \times A_a \times V_o \times C$, where $Org_s = Org \cup S, R_s = R \cup S, A_a = \mathcal{A} \cup \mathcal{A}$ and $V_o = V \cup O$. More precisely, if $auth$ is an organization or a subject, g is a role or a subject, t is a view or an object, p is an activity or an action and c is a context, then $Permission(auth, g, p, t, c)$ means that $auth$ grants permission to g to perform p on t in context c.

- ***Obligation*** is a predicate over domains $Org_s \times R_s \times A_a \times V_o \times C \times C$. More precisely, if $auth$ is an organization or a subject, g is a role or a subject, t is a view or an object, p is an activity or an action and c_1, c_2 are contexts, then $Obligation(auth, g, p, t, c_1, c_2)$ means that g is obliged to $auth$ to perform p on t when context c_1 is activated and before the activation of context c_2.

- Concrete predicates ***Is-Permitted*** and ***Is-Obliged*** are predicates over domains $S \times A \times O$. These predicates enable to specify permissions, prohibitions and obligations at the concrete level, which are based on subjects, actions and objects. A concrete predicate is derived from an abstract one when the associated context holds. More details are given in section 2.4.

- ***Is-Fulfilled*** and ***Is-Violated*** are predicates over domains $S \times A \times O$. If s is a subject, a is an action and o is an object, then $Is\text{-}Fulfilled(s, a, o)$ (resp. $Is\text{-}Violated(s, a, o)$) means that, subject s has fulfilled (resp. violated) the obligation to perform action a on object o. When the predicate *Is-Violated* is derived, a sanction is imposed. This generally corresponds to the activation of new security rules (e.g. prohibitions or obligations).

2.2 Obligations as duty objects

The administration model is based on an object-oriented approach [4]. Thus users having administrative privileges do not manipulate privileges directly (i.e. Permission, Prohibition and Obligation), but instead use (i.e. create, update or delete) objects having a specific semantics and belonging to specific views, called administrative views. Each object has an identifier that uniquely identifies the object and a set of attributes to describe it.

The view $Norm$ is used to specify and manage users' privileges. Therefore, objects belonging to this view may have three types: *license*, *ban* or *duty*; and according to this type the existence of an object in this view is interpreted as a Permission, a Prohibition or an Obligation, respectively. In [4], it is shown how to model license objets and how to use them to manage delegation of permissions. Duty objects have the following attributes: *Auth*: entity that imposes the duty, *Obligatee*: subject to which the obligation is assigned, *Task*: action that must be executed, *Target*: object on which the duty applies, *Context*: specific conditions that must be satisfied to activate the obligation and *Context_Violation*: specific conditions that represent the

obligation deadline. There is the following rule to derive obligations from duty objects[1]:

$$Obligation(Auth, R, A, V, Ctx, CtxV)\text{:-}$$
$$Use(Org, D, norm), Type(D, duty), Auth(D, Auth),$$
$$Obligatee(D, R), Task(D, A), Target(D, V),$$
$$Context(D, Ctx), Context\_Violation(D, CtxV).$$

When the attribute obligatee is a role we need an additional rule to derive obligations that apply to subjects[2]:

$$Obligation(Auth, S, A, O, C, C_V) \text{ :-}$$
$$Obligation(Auth, R, A, O, C, C_V),$$
$$Use(Auth, R, role), Empower(Auth, S, R).$$

Note that the attribute *Auth* can be an organization (e.g. a government, a hospital), or a subject. Thus, obligations can be imposed by a moral or a physical authority, respectively.

2.3 Group Obligations

The OrBAC model introduces the concept of group obligations [14]. It allows the expression of obligations which apply to a group of subjects (role) that share the related responsibility, or/and to a group of actions (activity) or objects (view) that express a set of alternative operations for the fulfillment of the obligation. A group obligation is specified using a new context Ctx_g called *group context*:

$$Obligation(Auth, R, A, V, Ctx, Ctx\_v, Ctx\_g).$$

where Ctx_g is an obligation group context defined as follows: $groupContext(Qe, Qe, Qe)$, Q is one of the quantifiers $\{\forall, \exists\}$ and e is an element of $\{a, s, o\}$.

An obligation is called group obligation when its group context contains the quantifier $\exists$, otherwise it is a regular obligation, i.e. an obligation for every subject empowered in the obligation's role to take every action considered in the obligation's activity on every object used in the obligation's view. An example of a group obligation is that "one of the meeting participants must submit a meeting report":

$$Obligation(Auth, meeting\_member, submit, report,$$
$$end\_meeting, deadline\_report, groupContext(\exists o, \exists a, \exists s)).$$

where contexts *end_meeting* and *deadline_report* are activation and violation contexts, and $groupContext(\exists o, \exists a, \exists s)$ means that *meeting_member* is a set of users that share the obligation to submit a report.

In the rest of this paper, we omit the group context in the case of regular obligations.

2.4 Obligations and Contexts

In our model, we are based on contextual security rules, this means that, the security rule does not apply statically but its activation may depend on contextual conditions (examples of context may be Night, Working-Hours or Urgency). Contexts are classified into five types: the *Temporal* context that depends on the time at which the subject is requesting for an access to the system, the *Spatial* context that depends on the subject location, the *User-declared* context that depends on the subject objective (or purpose), the *Prerequisite* context that depends on characteristics that join the subject, the action and the object, the *Provisional* context that depends on previous actions the subject has performed in the system. There is also a context called *Nominal* that is always active for any subject, action and object.

Conditions that must be satisfied to derive that a context is active are modeled by a logical rule called *context definition*. In our model, it is possible to define a state-based context using derivation rules as follows:

$$Hold(Org, S, A, O, Ctx)\text{:-}$$
$$p_1(X_1), \ldots, p_n(X_n).$$

This means that context Ctx holds in organization Org for subject S, action A and object O if a sequence of conditions denoted by the predicates $p_1(X_1)$, ..., $p_n(X_n)$ is true.

In addition, it is possible to define an event-based context [12] using ECA event definition [1] to take into account the dynamic activation of obligations:

$$Hold(Auth, S, A, O, Ctx) \textbf{ after } a(X)$$
$$\textbf{if } p_1(X_1), \ldots, p_n(X_n)$$

This is an event definition proposition meaning that context Ctx holds between *Auth*, S, A and O if the conditions $p_1(X_1)$, ..., $p_n(X_n)$ are true in the state following the execution of the action $a(X)$.

We can also combine these elementary contexts to define new composed contexts by using conjunction, disjunction and negation operators: &, $\oplus$ and $\bar{}$. This means that if c_1 and c_2 are two contexts, then c_1 & c_2 is a conjunctive context, $c_1 \oplus c_2$ is a disjunctive context and $\bar{c_1}$ is a negative context.

The notion of context is very useful in our model and allows us to specify dynamic security policy. In this section, we are interesting in contexts used to deal with obligations, namely event-based contexts. State-based contexts are used to deal with permissions and prohibitions.

Obligation Activation. Actual obligations are derived for some subject, action and object when the obligation context holds:

$$Hold(Auth, S, A, O, C)$$
$$\textbf{initiates } Is\_Obliged(S, A, O)$$
$$\textbf{if } Obligation(Auth, S, A, O, C, CtxV)$$

This is an active rule proposition meaning that the activation of the obligation context C between *Auth*, S, A and O triggers the derivation of a concrete obligation for subject S to fulfill action A on object O. When the obligation is fulfilled the predicate $Is\_Fulfilled$ is derived and the corresponding actual obligation is removed from the policy.

Obligation Violation. When the violation context is active and the obligation is not yet fulfilled, we need to derive a violation:

[1]To express rules and facts, we shall actually use a prolog-like notation. Terms starting with a capital letter, such as Subject, correspond to variables and terms starting with a lower case letter, such as peter, correspond to constants

[2]There are two other similar rules to respectively interpret duties when the attribute $Task$ is an activity and the attribute $Target$ is a view.

$Hold(Auth, S, A, O, CtxV)$
initiates $Violation(S, A, O)$
if $Obligation(Auth, S, A, O, C, CtxV), Is\_Obliged(S, A, O)$

This is an active rule proposition meaning that the activation of the violation context $CtxV$ between $Auth$, S, A and O triggers the derivation of a violation for S if the concrete obligation is not yet fulfilled.

3. NOTIFICATION CONTEXT

In this section, we introduce the concept of the pre-notification of obligation activation and violation. This means that, in addition to the notification sent to users when their obligations are activated or violated, we propose to pre-notify users that they must perform an obligation before some activation/violation context associated with the obligation occurs. In fact, users may not know their obligations and even if it is the case, they may not know when these obligations will be activated and/or violated. Therefore, pre-notification enables users to understand and to be aware of their duties, i.e. in which time, condition or after which action their obligations are activated and violated.

In our model, we deal with notification and pre-notification similarly using notification predicates as follows:

- $Notify_{Obl}$ is a predicate over domains $Org_s \times S \times A \times O \times C \times C$. If $auth$ is an organization or a subject, s is a subject, a is an action, o is an object, c and c' are contexts, then $Notify_{Obl}(auth, s, a, o, c, c')$ means that user s is notified (or pre-notified) that his/her obligation is active (or will be activated) in context c and must be fulfilled before the activation of context c'.
- $Notify_{Viol}$ is a predicate over domains $Org_s \times S \times A \times O \times C$. If $auth$ is an organization or a subject, s is a subject, a is an action, o is an object and c is a context, then $Notify_{Viol}(auth, s, a, o, c)$ means that s is notified (or pre-notified) that he/she has violated (or will violate) his/her obligation in context c.

To define pre-notification, we need to specify when the activation and the violation of an obligation must be pre-notified. For this purpose, we define a new type of context that we call *notification context* and we denote $C^{\sim}$. Thus, each context c can be associated with its notification context $c^{\sim}$ which defines when users must be notified of the activation of c. This is specified as follows:

- $notifContext$ is a predicate over domains $C \times C$. If c and $c^{\sim}$ are contexts, then $notifContext(c, c^{\sim})$ means that $c^{\sim}$ is the notification context of c.

For instance, the spatial context entering room r can be notified when the user is close to room r:

$$notifContext(entering\_Room(r), closeTo\_Room(r)).$$

Or, the temporal context day d can be notified one day before:

$$notifContext(day(d), day(d-1)).$$

Therefore, activation and violation contexts can be associated with notification contexts in order to specify when obligation activation and violation are pre-notified. Obviously, the security administrator will define the notification context only if it is necessary. For instance, if users must be notified before a given time interval in order to fulfill the requested task successfully (e.g. to attend a meeting) or before/after performing a given action to be aware of their future obligations (e.g. to accept a contract). Otherwise, pre-notification is not required and users are only notified when the activation or violation of an obligation occurs.

We define now these two rules to derive notification and pre-notification for obligation activation and violation:

$Hold(Auth, S, A, O, C^{\sim})$
initiates $Notify_{Obl}(Auth, S, A, O, C, C_V)$
if $Obligation(Auth, S, A, O, C, C_V), notifContext(C, C^{\sim})$

$Hold(Auth, S, A, O, C_V^{\sim})$
initiates $Notify_{Viol}(Auth, S, A, O, C_V)$
if $Obligation(Auth, S, A, O, C, C_V), Is\_Obliged(S, A, O)$, $notifContext(C_V, C_V^{\sim})$

This means that when context $C^{\sim}$ or $C_V^{\sim}$ holds, then users are pre-notified that their obligation will be activated or violated in a given context C or C_V, respectively. By default, the notification context takes the value of it's corresponding context. Thus, users are also notified when the obligation is activated and violated (i.e. when context C and C_V holds, respectively).

4. NEGOTIATING OBLIGATIONS

Users must be able to manage their obligations if they do not have sufficient resource, competence or time to fulfill the requested task or for the sake of efficiency. This is also useful in the case of conflicts between obligations, since an obligation can be modified (delayed or cancelled) in order to solve the conflict, as suggested in [9]. In [16] authors propose the concept of accountability that says that obligations are assigned only when users are able to fulfill them, i.e. users have sufficient privileges and resource to carry out the requested task. But, we argue that this is not sufficient to take into account users' requirements because the system is not always aware of their capabilities and/or availabilities. For instance, in the case of a conference reviewing system, a reviewer that is obliged to submit a paper review in a given deadline, has necessary sufficient privileges and resource to perform this obligation. But, he/she may need to delegate the paper review to a more expert colleague, or also to negotiate a deadline extension with the PC Chair.

In this section, we describe how users can negotiate their obligations with the responsible entity in order to adapt them, e.g. to extend the obligation violation context or the obligation activation, or also to reduce the task. For this purpose, we define two types of obligations: contractual and organizational, according to whether the obligation is negotiable or not, respectively.

To take into account the obligation type, we add an additional attribute to the duty objects, called Ctx_Nego: the negotiation context that specifies in which conditions users are allowed to negotiate their duties. In the case of organizational obligations this context is not specified. Otherwise, a permission to negotiate the obligation is derived for the obligatee as follows:

$Permission(Auth, S, negotiate, D, C_N)$:-
$Use(Org, D, norm), Type(D, duty), Auth(D, Auth)$,
$Obligatee(D, S), Ctx\_Nego(D, C_N)$.

This means that obligatee S is allowed to negotiate duty D with authority $Auth$ when context C_N holds. Using this context the administrator may specify complex negotiation conditions such that users are allowed to negotiate only some of the duty attributes, e.g. the violation or the activation context, or also users are allowed to negotiate their obligations only before their violation.

Before presenting how users negotiate their obligations with the authority in favor of whom the duty is contracted, we first introduce the built-in predicates that are used in agent communication.

- *Request* is a predicate over domains $S \times S \times A \times O$. This predicate is used when subject s requests another subject s' to perform action a on object o.
- *Approve*, *Disapprove* are predicates over domains $S \times S \times A \times O$. $Approve(s, s', a, o)$ (resp. $Disapprove$ (s, s', a, o)) is used when subject s approves (resp. disapproves) the request of subject s' to perform action a on object o.
- *Counter_Proposal* is a predicate over domains $S \times S \times A \times O \times O$. $Counter\_Proposal(s, s', a, o, o')$ is used when a subject s negotiates the request of subject s' to perform a on o and proposes object o' as a counter proposal.

For the sake of simplicity, we only focus here on obligations associated with physical entity, namely when the authority is a subject, and not an organization. Negotiating with moral entity involves more sophisticated concepts and will be discussed in future work. Indeed, when the authority is an organization, such as the government, we need to specify the physical entities in charge of the negotiation, and also the entities in charge of the sanction in the case of violation. In addition, it is possible to negotiate obligations between two moral entities, i.e. the obligatee is also an organization, such as a hospital. In this case, we have to specify the physical entities in charge of the negotiation between the hospital and the government.

The Negotiation Protocol. Negotiation can be established between an obligatee (U) and the authority (U') in order to adapt the obligation before or after its activation/ violation (see Fig.1). The obligatee U can send a negotiation request to the authority U' in order to negotiate duty D as follows:

$$Request(U, U', negotiate, D).$$

The authority U' can approve or disapprove the proposed duty, or further negotiate the obligation by sending a counter-proposal with another duty D':

$$Approve(U', U, negotiate, D).$$
$$Disapprove(U', U, negotiate, D).$$
$$Counter\_Proposal(U', U, negotiate, D, D').$$

The obligatee U in turn, can approve, disapprove or renegotiate the duty. Note that, using the duty objects it is possible to negotiate all obligation attributes. This means that authorized users can request to modify the task, the target, the activation context or/and the violation context. We may also negotiate the sanction associated with the obligation, but this aspect is not addressed in this paper.

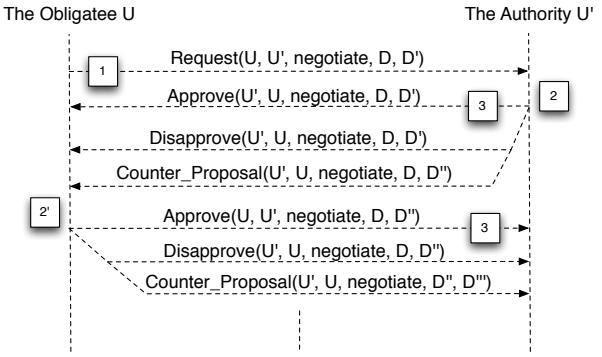

Figure 1: The Negotiation Protocol

Note that we specify the negotiation protocol using the security policy. This means that the protocol is established between the obligatee U and the authority U' only if the obligatee has the permission to negotiate duty D and the negotiation context (C_N) holds (step (1) in Fig. 1):

$$Is\_Permitted(U, negotiate, D).$$

The authority U' is obliged to answer to the negotiation request, and also the obligatee U must reply to the authority if he/she sends a counter_proposal (step (2) in Fig. 1). This is defined in our model by the following rule:

$$Obligation(auth, any\_subject, answer, duty, receive\_Req, answer\_Deadline, groupContext(\forall s, \forall o, \exists a)).$$

where *any_Subject* is a role in which all authorized users are empowered and *answer* is an activity which contains the actions $\{Approve, Disapprove, Counter\_\ Proposal\}$. As described in section 2.3, this is a group obligation, since users must take one of the actions considered in activity *Answer*. Context *receive_Req* is defined as follows:

$$Hold(Auth, S, Act, D, receive\_Req)$$
after $Request(\_, S, Act, D)$

Intuitively, this rule means that when a user receives a request, he/she is obliged to answer before a given deadline called here *answer_deadline* and that can be defined by the administrator. Similarly, users are obliged to answer when receiving a counter proposal (step (2') in Fig. 1). If the negotiation succeeds the authority U' is obliged to modify the duty according to the negotiated attributes (step (3) in Fig. 1). This is defined by the following obligation:

$$Obligation(auth, any\_Subject, modify, duty, nego\_Approval, approval\_Deadline).$$

where context *approval_Deadline* is a deadline fixed by the administrator to modify the negotiated duty and context *nego_Approval* is defined as follows:

$Hold(Auth, S, Act, D, nego\_Approval)$
after $Approve(S, \_, Act, D)$

5. DELEGATING OBLIGATIONS

Besides negotiation, users can delegate their obligations. Obviously, they must have the permission to delegate and in some cases they must have the agreement of the user who receives this delegation, i.e. the delegatee. More details about how to manage the delegation of obligations are given in [3] (namely, which users are allowed to delegate which obligations, to whom and in which context).

In our model, we distinguish between the delegation of an obligation to do some tasks and the delegation of the responsibility related to this obligation. As mentioned in [10], "in real life, the notion of obligation appears to include both interpretations, e.g. someone else may pay a traffic-offence fine for you, but may not go to jail for you". For this purpose, we specify three types of delegation according to whether the obligation is shared with full or limited responsibility or transferred [3]. We will further discuss the delegation of responsibility in future work.

Shared obligation with full responsibility. In this case the user, also called grantor, delegates his/her obligation and also the responsibility of this obligation to the delegatee. This means that, the grantor and the delegatee are obliged to perform the same task for the same authority. This can be defined in our model as a group obligation which applies to both the grantor and the delegatee:

$$Obligation(auth, deleg_G, a, o, c_{Act}, c_V, \\ groupContext(\exists o, \exists a, \exists s)).$$

where $deleg_G$ is a set of users containing the grantor and the delegatee, and $groupContext(\exists o, \exists a, \exists s)$ means that users in $deleg_G$ share the obligation to fulfill action a on object o when context c_{Act} holds and before the violation context c_V holds.

Shared obligation with limited responsibility. In this case the delegatee shares the obligation with the grantor, but does not share the responsibility with him/her. This means that the delegatee is obliged to the grantor and not to the original authority. The grantor remains obliged to the original authority:

$$Obligation(grantor, obligatee, a, o, c_{Act}, c_V).$$
$$Obligation(auth, grantor, a, o, c_{Act}, c_V).$$

Transfer. When the grantor transfers his/her obligation then he/she is neither obliged nor responsible for this obligation. Therefore, it is not allowed to transfer a limited responsibility like in delegation (i.e. only the delegatee is obliged to the original authority).

Note that the type of the delegated obligation may change according to the delegation type. When an organizational or contractual obligation is delegated with full responsibility or transferred, then its type is unchanged. But, if an organizational obligation is delegated with limited responsibility then we consider that the type of the delegated obligation is contractual and not organizational, because it is a new obligation created and managed by the grantor.

Delegation Notification. When a delegation is performed successfully the user who receives this delegation (i.e. the delegatee) must be informed that he/she has new obligations and responsibilities. For this purpose, we use the notification context $C^{\sim}$. Indeed, we can specify that the notification context of the obligation activation holds for the obligatee when the obligation is delegated to him/her. Thus, an *Obligation_Notification* will be derived to him/her as specified in section 3:

$Hold(Auth, S, A, O, C^{\sim})$
after $Create(G, D, norm\_delegation)$
if $Auth(D, Auth), Grantor(D, G), Obligatee(D, S),$
$Task(D, A), Target(D, O), Context(D, C),$
$NotifContext(C, C^{\sim})$

where *Norm_delegation* is a sub-view of the $Norm$ view defined to manage the delegation of obligations. Objects belonging to this view (i.e. duties) have the same attributes and semantics as duty objects belonging to the *Norm* view. They also have an additional attribute called *grantor*: subject who delegates the duty. Do not confuse attribute grantor with attribute authority, the grantor is the subject who delegates his/her obligation and the authority is the subject in favor of whom the duty is contracted. In [3] it is shown how to model duty objects.

As mentioned earlier, after the delegation of contractual obligations, the delegatee, in turn, can negotiate the obligation with the authority (which is the grantor in the case of delegation with limited responsibility) using the negotiation protocol. But before this, a delegation protocol can be established between the grantor and the delegatee in order to negotiate his/her consent. We investigate hereafter this protocol and how the consent can be requested dynamically.

5.1 The Delegation Protocol

In our model, we distinguish between two forms of delegation: bilateral agreement delegation (with delegatee's consent) and unilateral agreement (without consent request). In fact, according to the relationships between the grantor and the delegatee, the consent of this latter may be required or not to perform the delegation of obligation. We consider that there are two basic types of relationships: power relationships where one user is a subservient of another user and peer relationships where neither user is subservient to another user. In this case, the grantor is obliged to request the consent of the delegatee before delegating the obligation and/or the responsibility. To this end he/she must send a consent request: $Request(U, U', consent, D)$.
where D is the duty that the grantor U wants to delegate to the delegatee U'.

On receiving this request U' must send a response. He/she can approve, disapprove or negotiate the delegation: $Approve(U', U, consent, D), Disapprove(U', U, consent, D), Counter\_Proposal(U', U, consent, D, D')$.

Note that similarly to the negotiation protocol, the delegatee can negotiate all the attributes of the delegated duty, i.e. the task, the target, the activation context and the violation context. But also, he/she can negotiate the responsibility, this means that he/she can negotiate the authority value to specify that the obligation is shared with limited or with full responsibility. This protocol can be established, between a grantor and a delegatee in two cases: (1) in the case of peer relationships, the grantor is allowed to delegate a duty only if he/she requests the consent of the delegatee, or (2) in

the case of power relationships, the grantor is allowed to delegate a duty but can choose to request the delegatee's consent. In the next section, we present the first case and we discuss how we can deal with consent dynamically using the concept of pre-obligation contexts.

5.2 Dynamic Consent : Discussion

In the case of peer relationships, the grantor is obliged to get the consent of the delegatee before delegating to him/her the duty. For this purpose, we specify the following contextual rule:

$$Permission(org, any\_Subject, delegate, norm\_delegation, obligatee\_approval).$$

where *obligatee_approval* is a state-based context defined as follows:

$$Hold(Org, S, delegate, D, obligatee\_approval)\text{:-}\\ Approve(S', S, D), Grantor(D, S), Obligatee(D, S').$$

This means that user S is allowed to delegate obligation D if the delegatee S' has approved this delegation. This assumes that consent is approved before requesting delegation. In order to manage the consent dynamically, i.e. when the delegation is requested, we use the notion of pre-obligation contexts denoted C^*. This kind of context [13] is evaluated dynamically when the access is requested, in order to check if there is an unfulfilled pre-obligation actions that must be taken by the user before allowing the access. Thus, according to the context definition, pre-obligations are derived to the user. The fulfillment of these pre-obligations will activate the context. Therefore, to deal with the consent dynamically we can specify that context *obligatee_approval* is a dynamic context as follows:

$$Permission(org, any\_Subject, delegate, norm\_delegation, obligatee\_approval^*).$$

But, here the dynamic context definition must take into account that the action that must be fulfilled to activate the context does not concern the user who requests the delegation (i.e. the grantor), but the delegatee that must send an *Approve*. In this case, we may specify that the grantor must request the delegatee consent in order to activate the dynamic context. Therefore, context $obligatee\_approval^*$ will activate a pre-obligation for grantor S to request the consent of the delegatee S' to delegate obligation D as follows:

$$Obligation(S', S, req\_consent, D, immediate, C_v).$$

where context *immediate* is activated immediately after the delegation request and C_v is the violation context that can be specified in the dynamic context definition. Context $obligatee\_approval^*$ will be activated when grantor S requests the consent of the delegatee S' and S' sends his/her approval.

The concept of dynamic consent can also be used to specify other situations involving obligation delegation, such as the consent of the obligation authority. For instance, to delegate the review of a given paper to his/her assistant, the professor must request, in addition to the consent of the assistant, the consent of the authority of this obligation, i.e. the PC chair. Similarly as above, we can define pre-obligation contexts to deal with these situations. How dynamic contexts are defined and managed will be further discussed in future work.

6. RELATED WORK

In the literature [6, 15, 19, 20], many studies have been done on the modeling of obligations and some of them have considered the delegation of obligations, such as [10, 18, 23]. But, none of them have covered the topics discussed in this paper. To the best of our knowledge, neither the negotiation of obligations, the obligation pre-notification, nor the delegation negotiation have been addressed in role-based access control models. Negotiation has been studied in the case of collective obligations, such as in [7], where authors propose a negotiation protocol in which users propose a deal in order to distribute their collective tasks together with their associated sanctions. This is different from our protocol since our aim is to adapt individual obligations to the users' requirements and constraints. But, this is an interesting issue and we plan to further investigate it in our model, more precisely in the case of the delegation of group obligations.

In [16] authors propose the concept of accountability: "a system is said to be in an accountable state if all users have sufficient authorizations to carry out their obligations so long as every other user carries out his/her obligation". They also define an algorithm to determine which obligations will cause a system to become unaccountable. But, as we mentioned above, this is not sufficient to take into account users' requirements because the system is not always aware of their capabilities and/or availabilities. Moreover, they only define the accountability state but, they do not describe how obligations are managed to maintain this accountability, i.e. how the system adjust the obligations that cause the unaccountability. In our model, we give users some means to adjust their obligations using negotiation, delegation or transfer. In addition, the concept of obligation and violation notification can be a solution to help the system to adjust obligations and to keep the accountability state. For instance, when the system determines that an obligation will not be fulfilled, then instead of leaving the system to decide, a notification can be sent to the concerned user. In this case, we can add an additional information about the incapacity of the user to carry out the obligation, in order to incite him/her to adjust it.

The concept of consent has been explored in the context of privacy, like in [5, 21, 22, 2], and more precisely in healthcare systems when the patient consent is required to get an access to his/her medical and personal information. In [5] authors have dealt with consent as in our model, i.e. using pre-obligations. But to do so, they associate an obligation to each permission rule, whereas, we simply use pre-obligation contexts. In [21] it is suggested to delegate the consent to another user. This means that a patient can allow a doctor to further disclose his/her health data. We do not consider this aspect in this paper, but the issue of the propagation of delegation is discussed in [4], and it is possible to integrate this approach with our model to deal with consent propagation. In [22] the authors have proposed two types of consent: implicit consent with explicit deny policy and implicit deny with explicit consent policy. In this paper we have considered the second consent type. To deal with the first one, it is possible to specify that, for some situations (e.g. urgency), the consent context is always active. Thus,

the pre-obligation to request the consent will not be activated.

7. CONCLUSION

This paper has proposed a decentralized obligation management system where users are allowed to modify their duties according to their requirements. For this purpose, they can negotiate the obligation directly with the responsible entity (the authority), or negotiate its delegation with another user. Our main motivation has been to propose a more flexible access control model by considering the users requirements. Since, users can violate their obligations due to factors they do not always control, such as the existence of conflictual obligation, lack of time, of resource or of authorizations. These negotiations and delegations are carried out only under the security policy. Thus, the security administrator can control the propagation of duties by specifying which obligation is negotiable or/and delegatable and in which contexts (e.g. to whom, when, where, the consent is required or not, etc.). Negotiation and delegation succeeds only if the consent of the corresponding user is obtained. Thus, also users can control obligations that they are responsible for and obligations they receive by delegation. We have also introduced the concept of dynamic consent using contexts that dynamically activate a pre-obligation to request the consent, if it is not yet requested. Finally, we have proposed a new type of context, called notification context, in order to deal with the obligation notification. Notifications are sent to users before the activation or/and the violation of their obligations, or also when an obligation is delegated to them. This allows them to be aware of their duties.

In this paper, we have considered that consent is not persistent in time, therefore, users must request the consent each time it is required. As future work, it is possible to study persistent consent and more precisely the context of this kind of consent [17]. This means that users should be able to give their consent only for a given period of time or for a given purpose. In addition, they must be able to revoke their consent if they want. It is also possible to explore the concept of the delegation of consent as suggested in [21] and its revocation. We are confident that, using our delegation and revocation model, presented in [4], it is possible to deal with these concepts. Another future work will be to consider negotiation between moral authority and to include sanction negotiation. This needs to study the different types of responsibility [8] and to specify the physical entities in charge of the negotiation and of the sanction. In addition, we will further discuss the delegation of responsibility and we will address this issue in the context of the delegation of group obligations.

8. ACKNOWLEDGMENTS

This research has been supported by the ANR 07 SESUR FLUOR project.

9. REFERENCES

[1] C. Baral, J. Lobo, and G. Trajcevski. Formal Characterizations of Active Databases: Part II. In *DOOD*, 1997.

[2] M. Y. Becker and P. Sewell. Cassandra: Flexible Trust Management, Applied to Electronic Health Records. In *CSFW*, 2004.

[3] M. Ben-Ghorbel-Talbi, F. Cuppens, and N. Cuppens-Boulahia. An Extended Role-Based Access Control Model for Delegating Obligations. In *TrustBus*, 2009.

[4] M. Ben-Ghorbel-Talbi, F. Cuppens, N. Cuppens-Boulahia, and A. Bouhoula. A Delegation Model for Extended RBAC. *IJIS*, 9:3, 2010.

[5] E. Bertino, C. Brodie, S. Calo, L. Cranor, C.-M. Karat, J. Karat, N. Li, D. Lin, J. Lobo, Q. Ni, P. Rao, and X. Wang. Analysis of Privacy and Security Policies. *IBM Systems Journal*, 2009.

[6] C. Bettini, S. Jajodia, X. Wang, and D. Wijesekera. Obligation Monitoring in Policy Management. In *POLICY*, 2002.

[7] G. Boella and L. W. N. van der Torre. Negotiating the Distribution of Obligations with Sanctions among Autonomous Agents. In *ECAI*, 2004.

[8] L. Cholvy, F. Cuppens, and C. Saurel. Towards a Logical Formalization of Responsibility. In *Conference on Artificial Intelligence and Law*, 1997.

[9] J. Chomickiand and J. Lobo. Monitors for History-Based Policies. In *POLICY*, 2001.

[10] J. Cole, J. Derrick, Z. Milosevicand, and K. Raymond. Author Obliged to Submit Paper before 4 July: Policies in an Enterprise Specification. In *POLICY*, 2001.

[11] F. Cuppens and N. Cuppens-Boulahia. Modeling Contextual Security Policies. *IJIS*, 7(4), 2008.

[12] Y. El-Rakaiby, F. Cuppens, and N. Cuppens-Boulahia. From State-based to Event-based Contextual Security Policies. In *CMMSE*, 2009.

[13] Y. El-Rakaiby, F. Cuppens, and N. Cuppens-Boulahia. From Contextual Permission to Dynamic Pre-Obligation. In *ARES*, 2010.

[14] Y. El-Rakaiby, N. Cuppens-Boulahia, and F. Cuppens. Formalization and Management of Group Obligations. In *POLICY*, 2009.

[15] P. Gama and P. Ferreira. Obligation Policies: An Enforcement Platform. In *POLICY*, 2005.

[16] K. Irwin, T. Yu, and W. H. Winsborough. On the Modeling and Analysis of Obligations. In *CCS*, 2006.

[17] M. C. Mont, S. Pearson, G. Kounga, Y. Shen, and P. Bramhall. On the Management of Consent and Revocation in Enterprises: Setting the Context. Technical report, HP Laboratories, 2009.

[18] O. Pacheco and F. Santos. Delegation in a Role-Based Organization. In *DEON'*, 2004.

[19] J. Park and R. Sandhu. The UCON$_{ABC}$ Usage Control Model. *TISSEC*, 7(1), 2004.

[20] A. Pretschner, M. Hilty, and Basin. Distributed Usage Control. *Communications of the ACM*, 2006.

[21] C. Ruan, V. Varadharajan, and Y. Zhang. Delegatable Authorization Program and Its Application. In *SAM*, 2003.

[22] G. Russello, C. Dong, and N. Dulay. Consent-Based Workflows for Healthcare Management. In *POLICY*, 2008.

[23] A. Schaad and J. D. Moffett. Delegation of Obligations. In *POLICY*, 2002.